UOSLib – A Library for Analysis of Online-Learning Algorithms

Andreas Buschermöhle, Jens Hülsmann, Werner Brockmann

Universität Osnabrück
Albrechtstraße 28, 49069 Osnabrück
Tel. (0541) 969 2439
Fax (0541) 969 2480
E-Mail: Andreas.Buschermoehle@Uni-Osnabrueck.de,
Jens.Huelsmann@Uni- Osnabrueck.de,
Werner.Brockmann@Uni-Osnabrueck.de

1 Introduction

In many fields of application, the need for online-learning as an efficient and scalable machine learning technique increases. Either the amount of data is too big or continuously growing as in the emerging field of big data and data stream processing, or the application is time variant and thus requires a continuous adaptation to changing conditions. In all scenarios, the two most important learning tasks are *regression* and *classification*. Most algorithms for online-learning in these tasks are quite similar and can be readily used for both. A common algorithmic basis for both is often an approximation structure that is **l**inear **i**n the **p**arameters (LIP) for which a set of basis functions is fixed and only the parameter vector is adapted by learning. This has the advantage of fixed memory and computational effort which is of great importance either because of timeliness of the results, e.g. in embedded systems or streaming applications, or because of the huge amount of data to be processed otherwise. Different approximation structures, like fuzzy systems or polynomials belong to this category of LIP approximation structures.

To facilitate the development of new algorithms in this area and to systematically evaluate these methods, a suitable common framework is necessary. Such a framework must contain state of the art learning algorithms for comparison and be easily extensible with new ones. Furthermore it should be applicable to both tasks of regression and classification and should allow an easy setup of investigations with different properties of a learning scenario, e.g. degree of data noisiness, data linearity, or independence of

consecutive data. Just as well, the approximation structure must also be exchangeable to compare its impact on the resulting performance. This way, a systematic investigation of learning algorithms within the application is possible.

2 Online-Learning Libraries

Different libraries for the application of online-learning or learning of LIP approximations have been developed in recent years. One of the most widespread is MOA (Massive Online Analysis) [1] as a library for data stream mining. It is focused on stream classification, stream clustering and outlier detection, providing a variety of algorithms for each task. Yet, regression is not present in this library. Another big library for online-learning is Jubatus [2] which provides algorithms for fast online-learning with a focus on distributed systems. The library contains algorithms for the tasks of classification, regression, recommendation, graph mining, and anomaly detection, but only one learning algorithm for regression is present. Furthermore, Jubatus was mainly developed for distribution of learning tasks to several machines as to cope with high data rates and not to investigate an online-learning algorithm itself in a systematic way.

A library with tighter focus on the investigation of online-learning algorithms is OLL (Online-Learning Library) [3]. But with a focus on natural language processing this library likewise does not include the task of regression. Additionally, it contains only some basic algorithms and is not up to date. More learning algorithms but also with a focus on linear classification are available in LIBLINEAR [4]. But here the linear classification is not embedded into the setup of online learning, and again regression is not included. Similarly the most up-to-date library LIBOL (Library for Online Learning Algorithms) [5] includes many state of the art algorithms but again only for binary and here as well multi-class classification.

In conclusion, no library exists that connects online-learning for both tasks of regression and classification and combines it with tools for a systematic investigation. Especially the task of regression is severely underrepresented in available libraries. Furthermore, all libraries dealing with LIP approximations only use the simple linear model $y = \omega^T \cdot x$ where the output is just a linear combination of the input values. Thus no more elaborate approximation structure is used and the learning algorithms cannot be easily investigated in their interplay with the structure.

3 Scope of UOSLib

In this paper we hence present the *UOSLib* (Unified Online-learning Systems Library) as an extensible open source library of online-learning algorithms for Matlab. The online-learning task is characterized by learning on a sequence of data which can be described in *steps* (see Alg. 1) both for classification and regression. In step t the learning algorithm is presented an *instance* $x_t \in \mathbb{R}^d$ which is transferred from input space to parameter space by a fixed LIP approximation structure through a vector of *basis functions* $\phi(x_t) \in \mathbb{R}^n$, i.e. to allow more expressiveness than a simple linear approximation. This input is then used to predict its *label* $\hat{y}_t$ through a *model* $\hat{y}_t = f(\phi(x_t), \omega_t)$ with the *parameter vector* ω_t, which is the hypothesis maintained by the learning algorithm. Afterwards, the correct label y_t is given and the learning algorithm suffers an *instantaneous loss* $l(\hat{y}_t, y_t) \geq 0$ reflecting how wrong the prediction was. With the new pair of an instance and its corresponding label, henceforth called an *example* (x_t, y_t), the learning algorithm updates its hypothesis to ω_{t+1}.

Alg. 1: Stepwise Online-Learning

initialize parameter vector $\omega_1 \in \mathbb{R}^n$
initialize additional information Σ_1 //second order
for $t = 1, 2, \ldots, N_D$ **do**
  receive $x_t \in \mathbb{R}^d$
  predict its label $\hat{y}_t = f(\phi(x_t), \omega_t)$
  reveal true label y_t
  suffer loss $l(\hat{y}_t, y_t)$
  update parameter $\omega_{t+1} = \omega_t + \Delta_\omega(\omega_t, (x_t, y_t), \Sigma_t)$
  update information $\Sigma_{t+1} = \Sigma_t + \Delta_\Sigma(\omega_t, (x_t, y_t), \Sigma_t)$ //second order
end

Two classes of learning algorithms can be distinguished. On the one hand, *first order algorithms* update the parameter vector only based on the current parameter vector ω_t and the example (x_t, y_t). Thus they are memoryless and deal with every example the same way throughout the whole learning process. On the other hand, *second order algorithms* use additional fixed size information Σ_t to update the way of incorporating a new example as well (shown grayed in Alg. 1). For example, the recursive least squares method [6] incrementally tracks an estimate of the covariance matrix and uses it to further adjust the parameter update.

Depending on the learning task the evaluation of the output and the loss function differ. For regression the evaluation is given by $\hat{y}_t = f(\phi(x), \omega_t) =$

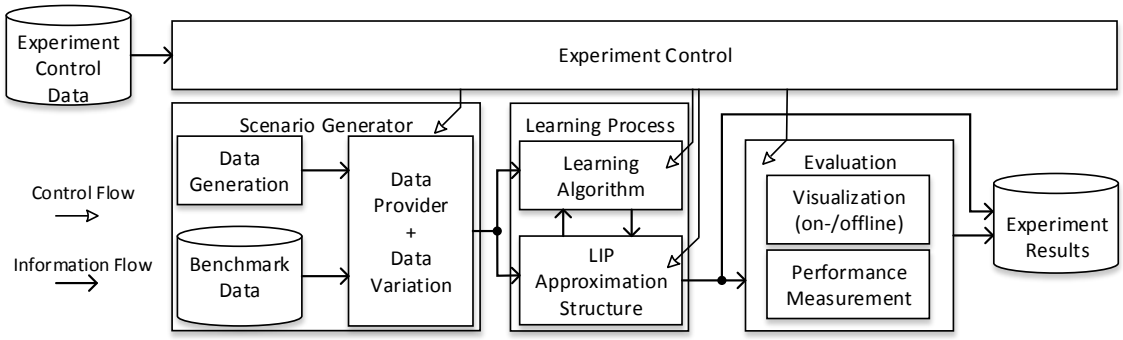


Fig. 1: Block diagram of the *UOSLib*-modules.

$\omega_t^T \cdot \phi(x)$ with $\hat{y}_t \in \mathbb{R}$. A commonly used loss function is the *squared loss*

$$l_r(\hat{y}_t, y_t) = (\hat{y}_t - y_t)^2 \,. \tag{1}$$

For classification the evaluation is typically given by $\hat{y}_t = f(\phi(x), \omega_t) = \text{sgn}(\omega_t^T \cdot \phi(x))$ with $\hat{y}_t \in \{-1, +1\}$. Here a commonly used loss function is the *hinge loss*

$$l_c = \begin{cases} 0 & \text{if } \hat{y}_t \cdot y_t \geq 1 \\ 1 - \hat{y}_t \cdot y_t & \text{otherwise} \,. \end{cases} \tag{2}$$

Within this learning scheme *UOSLib* provides a basis to compare a variety of approximation structures $\phi(x)$ and learning algorithms $\omega_t \rightarrow \omega_{t+1}$ together with a collection of different learning scenarios for both tasks.

4 Concept

To support the development of new learning algorithms with *UOSLib*, Matlab was chosen as a widespread basis on the one hand to allow for rapid prototyping and easy analysis with onboard tools and on the other hand because it is optimized for linear algebra operations which is the core of LIP approximations.

The concept of *UOSLib* is based on three aspects as shown in Fig. 1. First, a *scenario generator* can be set up to generate synthetic data sets in a reproducible way to test the algorithms in different situations based on the exactly same data. The scenario generator hence allows to adjust specific properties of the learning scenario like noise level on the target labels, data density, independence of the input data, etc. This way the influence of these properties on the different learning algorithms can be evaluated systematically. Through a fixed random seed, the complete setup is reproducible and

can be applied to different combinations of learning algorithm and approximation structure. Additionally, an interface to load (benchmark) datasets from a file is provided for an easy junction to external data sources. For external data some properties can be adjusted systematically as well, like added noise and order of presentation. All generated learning scenarios are easily mapped either to a regression task or to a binary classification task by taking the sign of the output. Most importantly, a setup can be uniquely specified by a footprint consisting of one tuple specifying the scenario and one the approximation structure, as shown in the following chapters, thus making it reproducible (by other researchers).

Second, for the learning process common interfaces to different approximation structures as well as learning algorithms are provided to easily exchange the structure or algorithm in use for comparison. This allows to test different combinations of

- learning task: regression or classification
- learning scenario: different properties of the data
- approximation structure: mapping input to parameter space $\mathbb{R}^d \rightarrow \mathbb{R}^n$
- learning algorithm: update of the parameter vector

by exchanging or systematically varying one of the four parts.

Third, the learning process is evaluated with different measures that are tracked over time. As online-learning is a continuous process, a cumulative performance measurement in a single number cannot present all relevant information. Rather, the progress of different measures over time is important to analyze the behavior. Therefor, *UOSLib* incorporates three measures that are updated after every learning step t. The cumulative loss l_c in Eq. (3) corresponds to the online performance, i.e. how well the predictive quality is at the respective step. In contrast to that, the mean data loss l_d in Eq. (4) corresponds to the quality on all examples seen up to the respective step, i.e. how well the general relationship of the examples was learned. Furthermore, for synthetic data it is possible to estimate the mean ground-truth loss l_g in Eq. (5), i.e. how well the learned approximation suits undisturbed and regularly sampled examples, thus covering the ability to generalize and cancel noise. To determine this, additional test examples $(\widetilde{x_i}, \widetilde{y_i})$ are directly drawn on a fine-grained regular grid covering the complete input space regardless of the density of training data and without any

disturbance.

$$l_c(t) = \sum_{\tau=0}^{t} l(f(\phi(x_\tau), \omega_\tau), y_\tau) \quad (3)$$

$$l_d(t) = \frac{1}{t} \sum_{\tau=0}^{t} l(f(\phi(x_\tau), \omega_t), y_\tau) \quad (4)$$

$$l_g(t) = \frac{1}{N_g} \sum_{i=0}^{N_g} l(f(\phi(\widetilde{x_i}), \omega_t), \widetilde{y_i}) \quad (5)$$

5 Main UOSLib Modules

To ease the use of the *UOSLib* functionality, a survey of its main functional units is given in the following with the relevant *UOSLib* function headers.

5.1 Scenario Generator

The scenario generator provides a set of examples which is used for training in a sequential order. All scenarios are scaled to enforce that the input values lie in the interval $[-10, +10]$ and the output is restricted to the interval $[-1, +1]$. By fixing the seed `rSeed` of the random number generator, the examples are fully reproducible. The generator is called through the function `icl_loadDS` which has the following interface:

```
[data groundTruth dim] =
  icl_loadDS(mode, func, ND, NG, noise, minPath)
```

The setup specifies the learning task with the parameter `mode` which is either *regression* or *classification*. The underlying function used to generate the examples is given by the string `func` that selects one of several predefined test functions (see Tab. 1) that differ in their amount of nonlinearity and changes of the monotonicity. If this string starts with `dataset`, it is interpreted as a path to a file from which the examples are loaded. Parameter `ND` specifies the number of examples to generate for learning and parameter `NG` the number of regularly sampled ground truth data per dimension for evaluation. The parameter `noise` sets the standard deviation of normally distributed noise on the training labels. Lastly, parameter `minPath` selects whether the examples are ordered randomly or in a sequential way, resembling a continuous movement of the samples taken within input space.

If parameter `minPath` is `true`, the randomly generated examples are ordered in such a way that, starting at the lowest value in each dimension, i.e. -10, the next value in the sequence is chosen from the remaining set of randomly drawn examples to have a minimal distance to the current value (see Fig. 2 for an example).

`func`	Description	Dim.
`linear`	straight line	1
`nonlin`	exponential function	1
`sine`	sine function	1
`linear2`	linear plane	2
`twocircles`	minimum distance to two corners	2
`crossedridge`	crossed ridge function	2
`spiral`	spiral loop (typical classification task)	2
`highdimlin`	linear hyperplane	20
`highdimnonlin`	squareroot hyperplane	20
`relearn`	3d-order-polynomial changing coefficients after half of training data	1
`dataset...`	If the string starts with `dataset`, the string will be interpreted as a relative path to a file. The file should contain arbitrary many columns of inputs and one last column of the target output with space-separation.	

Tab. 1: Different possible functions to draw data from.

The scenario generator function returns the examples in `data` as a two dimensional matrix with one example in each row and the ground-truth data for comparison in `groundTruth`. Depending on the function selected for generation, the dimensionality of the scenario differs which is returned in `dim`. All in all, the learning scenario hence can be uniquely identified by the following tuple:

(`mode`, `func`, `ND`, `NG`, `noise`, `minPath`, `rSeed`)

5.2 Approximation Structures

The approximation structure is determined by the vector of basis functions $\phi : \mathbb{R}^d \rightarrow \mathbb{R}^n$. Currently three different approximation structures are

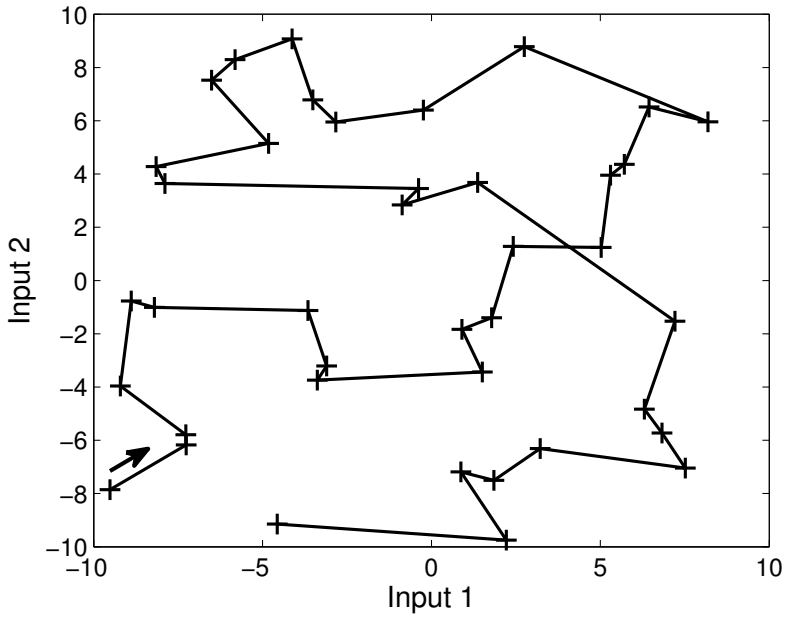


Fig. 2: Example of a randomly generated set of 40 examples with its according minimal path shown by the connections reflecting the order of presentation.

implemented. First, an additive polynomial of the form

$$f(x,\omega) = \omega_0 + \sum_{n=1}^{d} \sum_{m=1}^{N_o} \omega_{(n,m)} \cdot (x_n)^m \tag{6}$$

is given as an example for approximation structures with *globally* effective parameters by the function `icl_genPoly`. It produces an additive combination of polynomials of order N_o in every component x_n of the input vector. This approximation structure has the advantage of increasing only linearly in complexity with the scenario dimensionality at the cost of highly interacting parameters. The according function has the interface

```
ILS = icl_genPoly(order, dim)
```

receiving the polynomial order and the dimensionality. It returns a structure, containing the vectors of basis functions $\phi_{n,m}(x)$ and ω.

In contrast to that, two variants of a *local* approximation structure are implemented as grid-based lookup tables (GLT), one with an equally spaced grid (Fig. 3 left) and one with a spacing which can be set arbitrarily (Fig. 3 right). On such a grid in input space, the output is defined by the hight

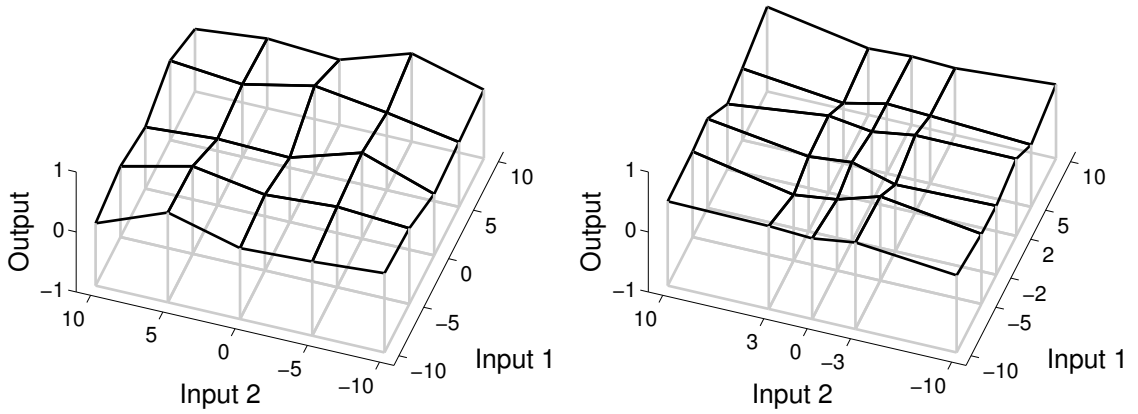


Fig. 3: Example of a grid-based lookup table with linear interpolation for two dimensions. The grid-points are either equally spaced, here with five points per dimension (left), or specified dimension-wise here at $\{-10, -5, -2, 0, 2, 5, 10\}$ for the first dimension and $\{-10, -3, 0, 3, 10\}$ for the second (right).

of the approximation at the grid-points and either a linear or a Gaussian interpolation in-between. Linear interpolation ensures total locality of the parameter influence, whereas Gaussian interpolation results in a smoother surface. The parameters of a GLT thus only have local influence on the output and hence do not interact as much. This causes a completely different learning behavior than global parameters. But with such a local structure, the curse of dimensionality leads to an exponentially increasing number of parameters with increasing dimensionality. For implementing this structure either `icl_genGLT` can be used with the interface

```
ILS = icl_genGLT(num, dim, base)
```

to get a regularly spaced grid in input space with the parameters `num` for the number of grid positions in each of the `dim` dimensions and `base` to choose between linear and Gaussian.

Alternatively, a more complex setup of a GLT structure is possible with the function `icl_genGLTarb` by defining an arbitrary grid structure through

```
ILS = icl_genGLTarb(loc, dim, base)
```

where an array of locations `loc` is specified containing a location array for each input dimension. For the example of Fig. 3 this array has the form $\{[-10, -5, -2, 0, 2, 5, 10]\ , [-10, -3, 0, 3, 10]\}$.

The used approximation structure is uniquely identified as well by:

(Poly, `order`) or (GLT, `num`, `base`) or (GLT, `loc`, `base`)

These approximation structures hence allow for an easy comparison of learning algorithms with globally or locally effective parameters. This distinction is especially significant for online-learning as a single example only presents local information for the parameter update.

5.3 Learning Algorithms

The main goal of *UOSLib* is to implement and to compare a variety of new and existing state of the art learning algorithms as well for regression as classification easily. On the one hand, they can be divided into the groups of *first* and *second order* learning algorithms. On the other hand, they are distinguished regarding their applicability to regression and/or classification. Currently the list of implemented learning algorithms contains the following.

First order algorithms are:

- Perceptron: Classical online-learning algorithm [7]
- PA: Passive-Aggressive algorithm in the three variants [8]:
  - PA: Parameter update fully aggressive
  - PA-I: Limited aggressiveness (linear slack variable)
  - PA-II: Limited aggressiveness (quadratic slack variable)
- IRMA: Incremental Risk Minimization Algorithm [9]

Second order algorithms are:

- CW: Confidence Weighted learning [10]
- AROW: Adaptive Regularization Of Weight vectors [11]
- GH: Gaussian Herding [12]
- RLS: Recursive Least Squares [6]

Except for IRMA, which is only applicable for regression tasks, and CW, which is only applicable for classification tasks, every algorithm can be used for both tasks.

To add a new learning algorithm, it must implement two interface functions. The first should provide algorithm specific initializations, e.g. the

covariance matrix of RLS. it therefor receives the structure ILS as it is generated by the approximation structure setup and the input dimensionality. Any information to be saved is changed within the returned ILS structure.

```
ILS = icl_initILS(ILS, dim)
```

The second function is used to update the ILS structure, i.e. its parameter vector and possible further information, with one single example incrementally. It receives again the ILS structure, as well as the example (x, y), the label yp predicted beforehand, and the mode reflecting whether the task is regression or classification.

```
ILS = icl_learn(ILS, x, y, yp, mode)
```

5.4 Evaluation

For evaluation of a learning run, the three measures of Eq. 3-5 are plotted over time to show their progress during learning. In Fig. 4 an example plot of such an evaluation is given for the comparison of PA and RLS on a simple scenario (mode = REG, func = sine, ND = 300, NG = 100, noise = 0.2, minPath = false, rSeed = 12345) with (GLT, num = 15, base = gauss) as the approximation structure. For one- and two-dimensional scenarios a visualization of the resulting approximation and the examples is supplied as well (see Fig. 5). This visualization can also be done live to follow the learning process with every example.

These evaluations help to reveal and compare the characteristic properties of the learning algorithms. In the example of Fig. 4 on a long run, RLS outperforms the predictive performance of PA with a slightly lower ground truth loss. But for low data densities, as in the beginning RLS tends to overfitting as the ground truth loss is high even though the data loss is low. Thus it is temporarily slightly outperformed by PA in the cumulative loss.

Similarly, classification tasks can be evaluated and visualized. In Fig. 6 the results of PA and AROW are compared for the learning scenario (mode = CLA, func = crossedridge, ND = 500, NG = 20, noise = 0.05, minPath = true, rSeed = 12345) with the approximation structure (GLT, num = 15, base = lin). The plot shows the given examples of the two classes with dots and crosses and additionally in white and gray the respective regions the learned approximation assigns to them. Apparently, both classifiers represent the basic structure of the data but they differ in some regions due to noise.

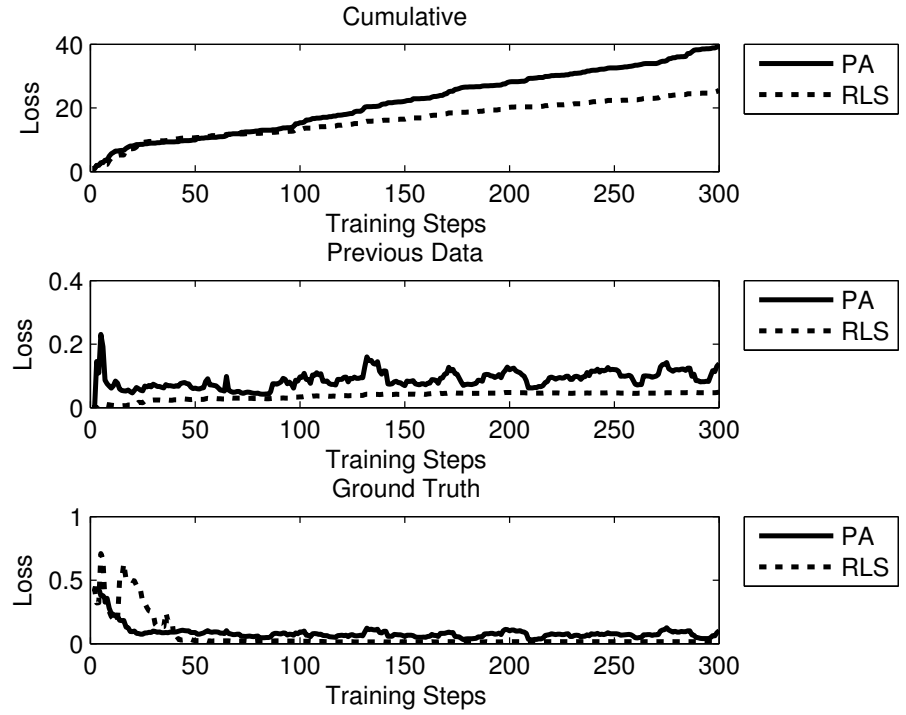


Fig. 4: Example plot of learning performance measures for a comparison of PA (solid line) and RLS (dashed line) on a simple regression task with a given GLT approximation structure.

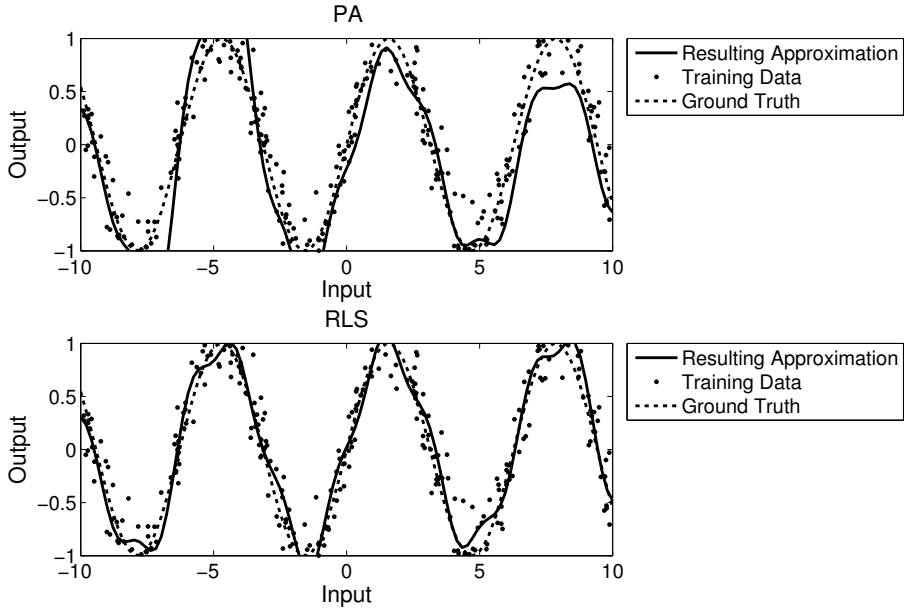
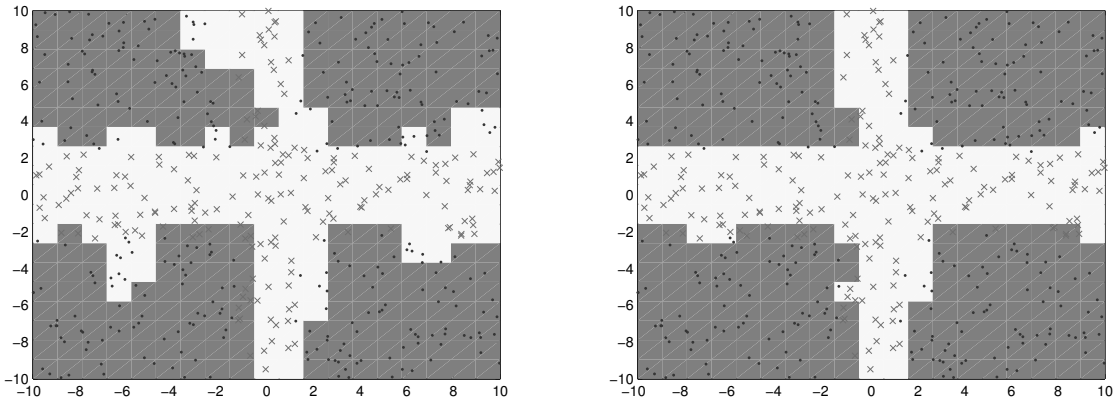


Fig. 5: Example plot of a resulting approximation for a comparison of PA (top) and RLS (bottom) on a simple one-dimensional regression task with a given GLT approximation structure.



(a) PA (b) AROW

Fig. 6: Resulting approximation for a comparison of PA (left) and AROW (right) on a simple binary classification task with a given GLT approximation structure. The examples for the classes are shown by dots and crosses and the resulting class regions in gray and white.

6 Conclusion

UOSLib is an open source library for systematic analysis of online-learning algorithms and online-learning tasks in general. It is easy to use and easy to extend. Based on Matlab, it consists of the four basic building blocks of scenario generator, approximation structure, learning algorithm, and evaluation. Each is easily extended with further algorithms and allows a systematic investigation of combinations of different instances for these building blocks.

The setup of an investigation is easily described through unique identifiers and thus very easy to reproduce, also by other researchers. *UOSLib* is thus well-suited for reference investigations in scientific publishing and may hence enhance on research standards.

Future work on the *UOSLib* covers three issues. First, a wider base of approximation structures and learning algorithms needs to be implemented, in order to include all state of the art approaches. Second, the scenario generator is a central feature. Its capability to set up different properties of a learning scenario allows solid research. Hence, further key features of generally relevant scenarios need to be identified and included in the generator. For easy use, they should be summarized into a scenario footprint which can be easily referenced. And third, for ease of use a more versatile graphical user interface would support wider accessibility of the library.

Literature

[1] Bifet, A.; Holmes, G.; Kirkby, R.; Pfahringer, B.: MOA: Massive Online Analysis. *Journal of Machine Learning Research* 99 (2010), S. 1601–1604.

URL `http://moa.cms.waikato.ac.nz`.

[2] Hido, S.: Jubatus: Real-Time and Highly-Scalable Machine Learning Platform. In: *Hadoop Summit*.

URL `http://jubat.us`. 2013.

[3] Okanohara, D.; Ohta, K.: Online Learning Library.

URL `http://code.google.com/p/oll/`. 2011.

[4] Fan, R.-E.; Chang, K.-W.; Hsieh, C.-J.; Wang, X.-R.; Lin, C.-J.: LIBLINEAR: A library for large linear classification. *Journal of Machine Learning Research* 9 (2008), S. 1871–1874.

URL `http://www.csie.ntu.edu.tw/~cjlin/liblinear/`.

[5] Hoi, S. C.; Wang, J.; Zhao, P.: *LIBOL: A Library for Online Learning Algorithms*. Nanyang Technological University.

URL `http://LIBOL.stevenhoi.org`. 2012.

[6] Blum, M.: Fixed Memory Least Squares Filters Using Recursion Methods. *IRE Trans. on Information Theory* 3 (1957) 3, S. 178–182.

[7] Rosenblatt, F.: The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain. *Psychological Review* 65 (1958) 6, S. 386–408.

[8] Crammer, K.; Dekel, O.; Keshet, J.; Shalev-Shwartz, S.; Singer, Y.: Online Passive-Aggressive Algorithms. *Journal of Machine Learning Research* 7 (2006), S. 551–585.

[9] Buschermoehle, A.; Schoenke, J.; Rosemann, N.; Brockmann, W.: The Incremental Risk Functional: Basics of a Novel Incremental Learning Approach. In: *Proc. of IEEE Int. Conf. on Systems Man and Cybernetics (SMC)*. IEEE Press. Accepted for publication. 2013.

[10] Dredze, M.; Crammer, K.; Pereira, F.: Confidence-Weighted Linear Classification. In: *Proc. of the 25th Int. Conf. on Machine Learning*, S. 264–271. ACM. 2008.

[11] Crammer, K.; Kulesza, A.; Dredze, M.; et al.: Adaptive Regularization of Weight Vectors. *Advances in Neural Information Processing Systems* 22 (2009), S. 414–422.

[12] Crammer, K.; Lee, D. D.: Learning via Gaussian Herding. *Advances in Neural Information Processing Systems* 23 (2010), S. 451–459.